Knowledge-Based Systems 107 (2016) 83–95

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Instance selection of linear complexity for big data

Álvar Arnaiz-González, José-Francisco Díez-Pastor, Juan J. Rodríguez, César García-Osorio*

University of Burgos, Spain

ARTICLE INFO

Article history:
Received 18 December 2015
Revised 3 April 2016
Accepted 30 May 2016
Available online 30 May 2016

Keywords:
Nearest neighbor
Data reduction
Instance selection
Hashing
Big data

ABSTRACT

Over recent decades, database sizes have grown considerably. Larger sizes present new challenges, because machine learning algorithms are not prepared to process such large volumes of information. Instance selection methods can alleviate this problem when the size of the data set is medium to large. However, even these methods face similar problems with very large-to-massive data sets.

In this paper, two new algorithms with linear complexity for instance selection purposes are presented. Both algorithms use *locality-sensitive hashing* to find similarities between instances. While the complexity of conventional methods (usually quadratic, $\mathcal{O}(n^2)$, or log-linear, $\mathcal{O}(n \log n)$) means that they are unable to process large-sized data sets, the new proposal shows competitive results in terms of accuracy. Even more remarkably, it shortens execution time, as the proposal manages to reduce complexity and make it linear with respect to the data set size. The new proposal has been compared with some of the best known instance selection methods for testing and has also been evaluated on large data sets (up to a million instances).

1. Introduction

The k nearest neighbor classifier (kNN) [11], despite its age, is still widely used in machine learning problems [9,17,20]. Its simplicity, straightforward implementation and good performance in many domains means that it is still in use, despite of some of its flaws [37]. The kNN algorithm is included in the family of instance based learning, in particular within the *lazy learners*, as it does not build a classification model but just stores all the training set [8]. Its classification rule is simple: for each new instance, assign the class according to the majority vote of its k nearest neighbors in the training set, if $k = 1$, the algorithm only takes the nearest neighbor into account [45]. This feature means that it requires a lot of memory and processing time in the classification phase [48]. Traditionally, two paths have been followed to speed up the process: either accelerate the calculation of the closest neighbors [3,4], or decrease training set size by strategically selecting only a small portion of instances or features [38].

Regarding the acceleration of algorithms, perhaps one of the most representative approaches is to approximate nearest neighbors, a broadly researched technique in which the nearest neighbor search is done over a sub-sample of the whole data set [56]. In this field, many algorithms have been proposed for approximate nearest neighbor problems [3,4,30,34,39].

The focus of this paper is on the second path, the reduction of data set size. The reason is that this reduction is beneficial for most methods rather than only those based on nearest neighbors. Although we will only consider the reduction of instances (instance selection) in this paper, the reduction could also be applied to attributes (feature selection), or even both at the same time [51]. The problem is that the fastest conventional instance selection algorithms have a computational complexity of at least $\mathcal{O}(n \log n)$ and others are of even greater complexity.

The need for rapid methods for instance selection is even more relevant nowadays, given the growing sizes of data sets in all fields of machine learning applications (such as medicine, marketing or finance [43]), and the fact that the most commonly used data mining algorithms for any data mining task were developed when the common databases contained at most a few thousands of records. Currently, millions of records are the most common scenario. So, most data mining algorithms find many serious difficulties in their application. Thus, a new term has emerged, “Big Data”, in reference to those data sets that, by volume, variability and speed, make the application of classical algorithms difficult [44]. With regard to instance selection, the solutions that have appeared so far to deal with big data problems adopt the ‘divide and conquer’ approach [13,22]. The algorithms proposed in the present paper offer a different approach, just a sequential but very quick and simple processing of each instance in the data set.

* Corresponding author.Fax: +34947258910.
E-mail addresses: alvarag@ubu.es (Á. Arnaiz-González), jfdpastor@ubu.es (J.-F. Díez-Pastor), jjrodriguez@ubu.es (J.J. Rodríguez), cgosorio@ubu.es (C. García-Osorio).

http://dx.doi.org/10.1016/j.knosys.2016.05.056

In particular, the major contribution of this paper is the use of Locality-Sensitive Hashing (LSH) to design two new algorithms, which offers two main advantages:

- Linear complexity: the use of LSH means a dramatic reduction in the execution time of the instance selection process. Moreover, these methods are able to deal with huge data sets due to their linear complexity.
- *On-the-fly* processing: one of the new methods is able to tackle the instances in one step. It is not necessary for all instances fit in memory: a characteristic that offers a remarkable advantage in relation to big data.

The paper is organized as follows: Section 2 presents the reduction techniques background, with special emphasis on the instance selection methods used in the experimental validation; Section 3 introduces the concept of *locality-sensitive hashing*, the basis of the proposed methods which are presented in Section 4; Section 5 presents and analyzes the results of the experiments and, finally, Sections 6 and 7 set out the conclusions and future research, respectively.

2. Reduction techniques

Available data sets are progressively becoming larger in size. As a consequence, many systems have difficulties processing such data sets to obtain exploitable knowledge [23]. The high execution times and storage requirements of the current classification algorithms make them unusable when dealing with these huge data sets [28]. These problems can be decisive, if a lazy learning algorithm such as the nearest neighbor rule is used, and can even prevent results from being obtained. However, reducing the size of the data set by selecting a representative subset has two main advantages: it reduces the memory required to store the data and it accelerates the classification algorithms [19].

In the scientific literature, the term "reduction techniques" includes [61]: prototype generation [32]; prototype selection [52] (when the classifier is based on kNN); and (for other classifiers) instance selection [8]. While prototype generation replaces the original instances with new artificial ones, instance selection and prototype selection attempt to find a representative subset of the initial training set that does not lessen the predictive power of the algorithms trained with such a subset [45]. In the paper, prototype generation is not addressed, however a complete review on it can be found in [57].

2.1. Instance selection

The aforementioned term "instance selection" brings together different procedures and algorithms that target the selection of a representative subset of the initial training set. There are numerous instance selection methods for classification, a complete review of which may be found in [21]. Instance selection has also been applied to both regression [2,33] and time series prediction [26,55].

According to the order in which instances are processed, instance selection methods can be classified into five categories [21]. If they begin with an empty set and they add instances to the selected subset, by means of analyzing the instances in the training set, they are called incremental. The decremental methods, on the contrary, start with the original training data set and they remove those instances that are considered superfluous or unnecessary. Batch methods are those in which no instance is removed until all of them have been analyzed, instances are simply marked from removal if the algorithm determines that they are not needed, and at the end of the process only the unmarked instances are kept. Mixed algorithms start with a preselected set of instances. The process then decides either to add or to delete the instances. Finally, fixed methods are a sub-family of mixed ones, in which the number of additions and removals are the same. This approach allows them to maintain a fixed number of instances (more frequent in prototype generation).

Table 1
Summary of state-of-the-art instance selection methods used in the experimental setup (taxonomy from [21]; computational complexity from [31] and authors' papers).

Strategy	Direction	Algorithm	Complexity	Year	Reference
Condensation	Incremental	CNN	$\mathcal{O}(n^3)$	1968	[27]
	Incremental	PSC	$\mathcal{O}(n \log n)$	2010	[46]
	Decremental	RNN	$\mathcal{O}(n^3)$	1972	[25]
	Decremental	MSS	$\mathcal{O}(n^2)$	2002	[6]
Hybrid	Decremental	DROP1-5	$\mathcal{O}(n^3)$	2000	[60]
	Batch	ICF	$\mathcal{O}(n^2)$	2002	[8]
	Batch	HMN-EI	$\mathcal{O}(n^2)$	2008	[41]
	Batch	LSBo	$\mathcal{O}(n^2)$	2015	[37]

Considering the type of selection, three categories may be distinguished. This criterion is mainly correlated with the points that they remove: either border points, central points, or otherwise. Condensation techniques try to retain border points. Their underlying idea is that internal points do not affect classification, because the boundaries between classes are the keystone of the classification process. Edition methods may be considered the opposite of condensation techniques, as their aim is to remove those instances that are not well-classified by their nearest neighbors. The edition process achieves smoother boundaries as well as noise removal. In the middle of those approaches are hybrid algorithms, which try to maintain or even to increase the accuracy capability of the data set, by removing both: internal and border points [21].

Evolutionary approaches for instance selection have shown remarkable results in both reduction and accuracy. A complete survey of them can be found in [16]. However, the main limitation of those methods is their computational complexity [36]. This drawback is the reason why they are not taken into account in this study, because the methods it proposes are oriented towards large data sets.

In the remaining part of this section, we give further details of the most representative methods used in the experimental setup. A summary of the methods considered in the study can be seen in Table 1.

2.1.1. Condensation

The algorithm of Hart, *Condensed Nearest Neighbor* (CNN) [27] is considered the first formal proposal of instance selection for the nearest neighbor rule. The concept of training set consistency is important in this algorithm and is defined as follows: given a non empty set X ($X \neq \varnothing$), a subset S of X ($S \subseteq X$) is consistent with respect to X if, using the subset S as training set, the nearest neighbor rule can correctly classify all instances in X. Following this definition of consistency, if we consider the set X as the training set, a condensed subset should have the properties of being consistent and, ideally, smaller than X. After CNN appeared, other condensation methods emerged with the aim of decreasing the size of the condensed data set, e.g.: Reduced Nearest Neighbor (RNN) [25]. One of the latest is the Prototype Selection by Clustering (PSC) [46], which uses clustering to speed up the selection process. So, the use of clustering gives a high efficiency to PSC, if compared against state-of-the-art methods, and better accuracy than other clustering-based methods such as CLU [40].

In [6], the authors proposed a modification to the definition of a selective subset [54], for a better approximation to decision borders. The selective subset can be thought of as similar to the idea of the condensed algorithm of Hart, but applying a condi-

tion stronger than the condition of consistency. The aim is to find the selected instances in an easier way, which is less sensitive to the random initialization of S and the order of exploration of X in Harts' algorithm. The subset obtained in this way is called the selective subset (SS).

A subset S of the training set X is a *selective* subset (SS), if it satisfies the following conditions:

1. S is consistent (as in Harts' algorithm).
2. All instances in the original training set, X, are closer to a *selective neighbor* (a member of S) of the same class than to any instance of a different class in X.

Then, the authors present a greedy algorithm which attempts to find selective instances starting with those instances of the training set that are close to the decision boundary of the nearest neighbor classifier. The algorithm presented is an efficient alternative to the Selective algorithm [54] and it is usually able to select better instances (the ones closer to the boundaries).

2.1.2. Hybrid

One problem that arises when condensation methods are used is their noise sensitivity, while hybrid methods have specific mechanisms to make them more robust to noise [37].

The DROP (Decremental Reduction Optimization Procedure) [60] family of algorithms comprises some of the best instance selection methods for classification [8,47,50]. The instance removal criteria is based on two relations: *associates* and nearest neighbors. The relation of associate is the inverse of nearest neighbors: those instances **p** that have **q** as one of their nearest neighbors are called associates of **q**. The set of nearest neighbors of one instance is called the *neighborhood* of the instance. For all instances, its list of associates is a list with all instances that have that particular instance in their neighborhood.

Marchiori proposed a new graph-based representation of the data set called Hit Miss Networks (HMN) [41]. The graph has a directed edge from each instance to its nearest neighbor on the different classes, with one edge per class. The information in the graph was used to define three new hybrid algorithms: HMN-E, HMN-C and HMN-EI. A couple of years later, HMNs were used to define a new information-theoretic instance scoring method to define a new instance selection method called Class Conditional Nearest Neighbor (CCIS) [42]. According to [21], HMN-EI is able to achieve more accurate data sets than CCIS which is the reason why HMN-EI was used in experimental setup.

The local-set (LS) concept, proposed for the very first time in [7], is a powerful tool for some machine learning tasks, including instance selection. A local-set of an instance **x** contains all those instances which are closer to **x** than its nearest neighbor of different class, its *nearest enemy*. The selection rule of the Iterative Case Filtering algorithm (ICF) [7] uses local sets to build two sets: *coverage* and *reachability*. These two concepts are closely related to the neighborhood and associate list used in DROP algorithms. The coverage of an instance is its LS, that can be seen as a neighborhood of the instance that, instead of considering a fixed number k of neighbor, includes all instances closer to the instance than its closest enemy. The reachable set of an instance is its set of associates. The coverage set of an instance is its neighborhood. The deletion rule is as follows: an instance is removed from the data set if its reachable set (its set of associates) is bigger than its coverage (its 'neighborhood'). This rule means that the algorithm removes an instance if other object exists that generalizes its information. To address the problem of noisy data sets, both ICF and DROP3, begin with a noise-filter stage. Recently, Leyva et al [37] presented three new instance selection methods based on LSs. Their hybrid approach, which offers a good balance between reduction and accuracy, is called Local Set Border Selector (LSBo) and it uses a heuristic criterion: the instances in the boundaries between classes tend to have greater LSs. As is usual in hybrid methods, LSBo starts with a noise filtering algorithm which was presented in the same paper called LSSm.

2.2. Scaling up instance selection

The main drawback of instance selection methods is their complexity that is quadratic $\mathcal{O}(n^2)$, where n is the number of instances [22] or, at best, log-linear $\mathcal{O}(n \log n)$; thus, the majority of them are not applicable in data sets with hundreds or even many thousands of instances [15]. Table 1 summarizes the computational complexity of the instance selection methods used in the experimental section.

One approach to deal with massive data sets, is to divide the original problem into smaller subsets of instances; known as stratification. The underlying idea of these methods is to split the original data set into disjointed subsets, then an instance selection algorithm is applied to each subset [10,13,22,24]. This approach is used in [22] where a method was proposed that addressed the splitting process, using Grand Tour [5] theory, to achieve linear complexity.

The problem known as big data refers to the challenges and difficulties that arise when huge amounts of data are processed. One way to accelerate instance selection methods and to be able to cope with massive data sets is adapt them to parallel environments [49]. To do so, the way that algorithms work has to be redesigned. The MapReduce paradigm offers a robust framework with which to process huge data sets over clusters of machines. Following up on this idea, a new proposal was presented recently by Triguero et al. [58].

3. Locality-sensitive hashing

The *locality-sensitive hashing* (*LSH*) is an efficient method for checking similarity between elements. It makes a particular use of *hash* functions that, unlike those used in other applications of hashing,[1] seeks to allocate similar items to the same bucket with a high probability, and at the same time to greatly reduce the probability of assigning dissimilar items to the same bucket [35].

LSH use is common to increase the efficiency of nearest neighbors calculation [3,21]. An indirect benefit of LSH for instance selection algorithms is the speeding up of nearest neighbor calculation, required in most of these sorts of algorithms. However, the complexity of the algorithms remain unchanged, since the loop nesting and structures of the algorithms remain the same. It is only the kNN step that is improved.

What we propose in this paper is a novel use of LSH, not merely as support for the calculation of nearest neighbors, but as an operation that defines the nature of the new instances selection algorithm. Basically, the idea is to make the instance selection on each of the buckets that will be obtained by LSH when applied to all instances. This process permits the selection of instances using a unique processing loop of the data set, thereby giving it linear complexity. So a reasonable question arises; when a classifier is trained with a selected subset obtained by this approach, will its prediction capabilities decrease? This article offers an experimental response to this question[2] But before giving the details of our

[1] The aim of conventional cryptographic hash functions is to avoid the collision of items in the same bucket.

[2] In any case, note that even a certain degradation of classifier performance would be acceptable, if the new algorithm achieved a substantial acceleration or reduction in storage [9], it is often better to gain a quick approximation within a reasonable time than an optimal solution when it is too late to use it.

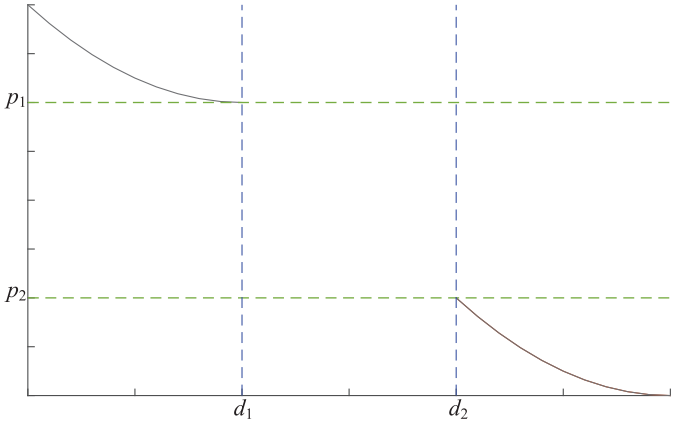


Fig. 1. Expected behavior of the probabilities of a (d_1, d_2, p_1, p_2)–sensitive function. The function will assign the same value to two instances with a probability greater than p_1, if their distance is shorter than d_1. The function will assign the same value to two instances with a probability lower than p_2, if their distance is greater than d_2. For distances between d_2 and d_1, there is no restriction regarding that the values the function can assign to the instances.

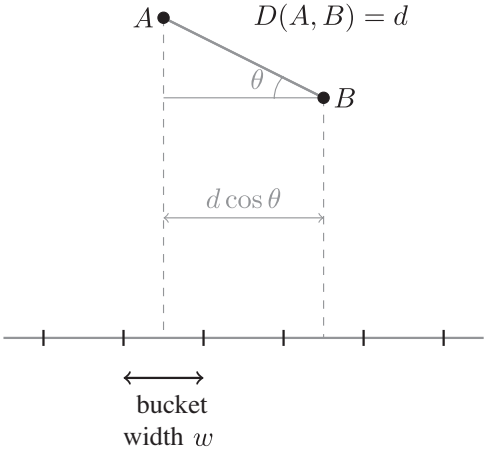


Fig. 2. Two points (A, B) at distance $d \gg w$ have a small chance of being hashed to the same bucket.

proposal, let us look at a brief introduction to the underlying theory of LSH.

3.1. Locality-sensitive functions

In this section we follow [35], to formally define the concept of local sensitivity and the process of amplifying a locality-sensitive family of functions.

Given a set of objects S and a distance measure D, a family of hash functions $\mathcal{H} = \{h : S \to U\}$ is said to be (d_1, d_2, p_1, p_2)–sensitive, if the following properties hold for all functions of h in the family $\mathcal{H}$:

- For all x, y in S, if $D(x, y) \leq d_1$, then the probability that $h(x) = h(y)$ is at least p_1.
- For all x, y in S, if $D(x, y) > d_2$, then the probability that $h(x) = h(y)$ is at most p_2.

In this definition, nothing refers to what happens when the distance of the objects is between d_1 and d_2 (see the representation in Fig. 1). However, distances d_1 and d_2 can be as close as possible, but the cost will be that p_1 and p_2 are also closer. However, as shown below, it is possible to combine families of hash functions that separate the probabilities p_1 and p_2 without modifying the distances d_1 and d_2.

Given a (d_1, d_2, p_1, p_2)–sensitive family of hash functions $\mathcal{H}$, it is possible to obtain a new family $\mathcal{H}'$ using the following amplification operations

AND-construction The functions h in $\mathcal{H}'$ are obtained by combining a fixed number r of functions $\{h_1, h_2, \ldots, h_r\}$ in $\mathcal{H}$. Now, $h(x) = h(y)$, if and only if $h_i(x) = h_i(y)$ **for all** i. If the independence of functions in $\mathcal{H}$ can be guaranteed, the new family of functions $\mathcal{H}'$ will be $(d_1, d_2, (p_1)^r, (p_2)^r)$–sensitive.

OR-construction The functions h in $\mathcal{H}'$ are obtained by combining a fixed number b of functions $\{h_1, h_2, \ldots, h_b\}$ in $\mathcal{H}$. Now, $h(x) = h(y)$, if and only if $h_i(x) = h_i(y)$ **for any** i. If the independence of functions in $\mathcal{H}$ can be guaranteed, the new family of functions $\mathcal{H}'$ will be $(d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)$–sensitive.

The AND-construction decreases the probabilities and the OR-construction increases them. However, if r and b are properly chosen and with the chaining of constructions the probability p_1 may be brought closer to 1, while the probability p_2 will stay reasonably close to 0.

In the experimental setup, the hash functions in the base family were obtained using the following equation [12].

$$h_{\vec{a},b}(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{w} \right\rfloor \tag{1}$$

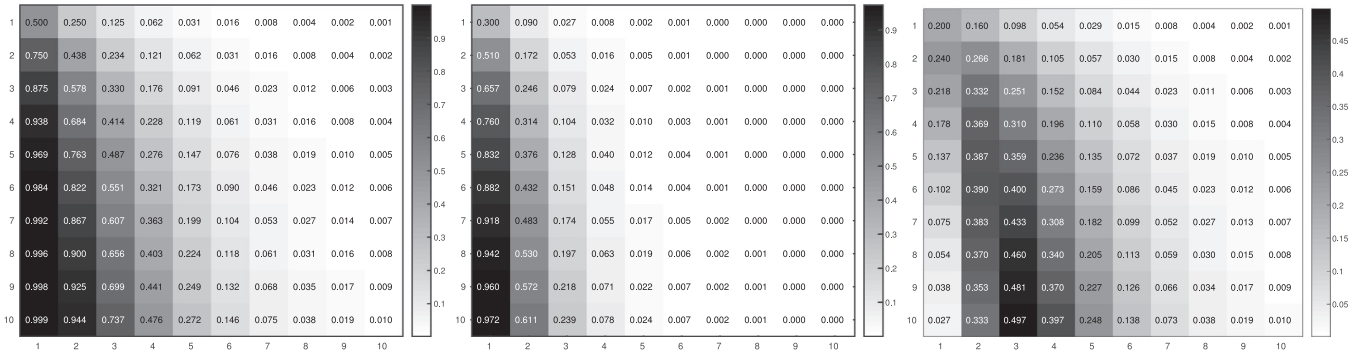
where $\vec{a}$ is a random vector (Gaussian distribution with mean 0 and standard deviation 1), b is a random real value from the interval $[0, w]$ and w is the width of each bucket in the hash table.

This equation gives a $(w/2, 2w, 1/2, 1/3)$-sensitive family. The reason for these numbers is as follows (suppose, for simplicity, a 2-dimensional Euclidean space), if the distance d between two points is exactly $w/2$ (half the width of the buckets) the smallest probability for the two points falling in the same segment would happen for $\theta = 0$, and in this case the probability would be 0.5, since d is exactly $w/2$. For angles greater than 0, this probability will be even higher; in fact, it will be 1 for $\theta = 90$. And for shorter distances than $w/2$, the probability will equally increase. So the lower boundary for this probability is 1/2. If the distance d is exactly $2w$ (twice the width of the bucket), the only chance for both points to fall in the same bucket is that their distances, once projected in the segment, are lower than w, what means that $\cos\theta$ must be lower than 0.5, since the projected distance is $d\cos\theta$ and d is exactly $2w$. For θ in the interval 0 to 60, $\cos\theta$ is greater than 0.5, so the only chance of $\cos\theta$ being lower than 0.5 is that θ is in the interval [60, 90], and the chance of that happening is at most 1/3. For distances greater than $2w$, the probabilities are even lower. So the upper boundary of this probability is 1/3. This reasoning is reflected in Fig. 2.

By using the $(w/2, 2w, 1/2, 1/3)$-sensitive family previously described, we have computed the probabilities p_1 and p_2 for the AND-OR construction with a number of functions from 1 to 10. Fig. 3 shows the probabilities p_1 (a) and p_2 (b) for the case of the chaining of an OR-construction just after an AND-construction, and the difference between these two probabilities (c). The row number indicates the number of functions used in the AND-construction, while the column number indicates the number of functions used in the OR-construction.

4. New instance selection algorithms based on hashing

This section presents the algorithms proposed in this work: LSH-IS-S and LSH-IS-F. The first completes the selection process in a single pass, analyzing each instance consecutively. It processes



(a) p_1 (b) p_2 (c) $p_1 - p_2$

Fig. 3. Probabilities p_1 and p_2 and the difference between them. The darker the color the higher the value. Each cell gives the value for the chaining of an OR-construction (number of combined basic functions on the x-axis) after an AND-construction (on the y-axis the number of combined functions).

instances in one pass, so not all instances need to fit in memory. The second performs two passes: in the initial one, it counts the instances in each bucket, in the second, it completes the instance selection with this information. The complexity of both algorithms is linear, $\mathcal{O}(n)$ (note that this is even true for the second algorithm, i.e. although two passes are performed).

Both algorithms can be seen as incremental methods, due to the fact that the selected data set is formed by successive additions to the empty set. However, the second one conforms more closely to batch processing because it analyzes the impact of the removal on the whole data set.

The main advantage of the presented methods is the drastic reduction in execution time. The experimental results show a significant difference when they are compared against state-of-the-art instance selection algorithms.

Algorithm 1: LSH-IS-S – Instance selection algorithm by hashing in one pass processing.

Input: A training set $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, set $\mathcal{G}$ of hash function families
Output: The set of selected instances $S \subseteq X$

1 $S = \varnothing$
2 **foreach** *instance* $\mathbf{x} \in X$ **do**
3 **foreach** *function family* $g \in \mathcal{G}$ **do**
4 $u \leftarrow$ bucket assigned to $\mathbf{x}$ by family g
5 **if** *there is no other instance of the same class of* $\mathbf{x}$ *in* u **then**
6 Add $\mathbf{x}$ to S
7 Add $\mathbf{x}$ to u
8 **return** S

4.1. LSH-IS-S: one-pass processing

As shown in Algorithm 1, the inputs of the LSH-IS-S method are: a set of instances to select and a set of families of hash functions. The loop processes each instance $\mathbf{x}$ of X, using the function families to determine the bucket u to which the instance belongs .[3] If in the bucket u assigned to the instance there is no other instance of the same class of **x,** $\mathbf{x}$ is selected and added to S and to the bucket u. The algorithm ends when all instances in X have been processed. Note that each instance is processed only once, which grants an extremely fast performance at the expense of not analyzing the instances that are selected in each bucket in detail. Instances are analyzed in sequence without needing information on other instances. This process means that the method may be used in a single-pass process, without requiring the whole data set to fit in the memory.

4.2. LSH-IS-F: a more informed selection

The algorithm explained in the previous section is remarkably fast and allows instances to be processed as they arrive, in one pass. On the other hand, because of how it works, it is not using all information that may be relevant to decide which instances to choose. For example, the algorithm has no control over the number of instances of each class that go to each bucket, because once an instance of a class is selected, it discards other instances of the same class that may come later.

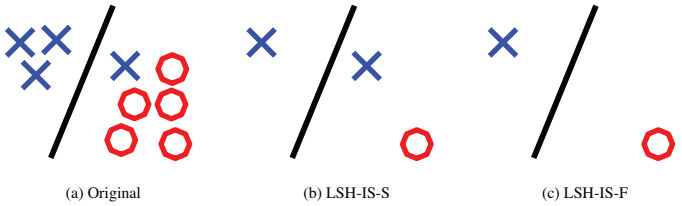
Algorithm 2: LSH-IS-F – Instance selection algorithm by hashing with two passes.

Input: A training set $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, set $\mathcal{G}$ of hash function families
Output: The set of selected instances $S \subseteq X$

1 $S = \varnothing$
2 **foreach** *instance* $\mathbf{x} \in X$ **do**
3 **foreach** *function family* $g \in \mathcal{G}$ **do**
4 $u \leftarrow$ bucket assigned to $\mathbf{x}$ by family g ;
5 Add $\mathbf{x}$ to u ;
6 **foreach** *function family* $g \in \mathcal{G}$ **do**
7 **foreach** *bucket* u *of* g **do**
8 **foreach** *class* y *with some instance in* u **do**
9 $I_y \leftarrow$ all instances of class y in u;
10 **if** $|I_y| > 1$ **then**
11 Add to S one random instance of I_y ;
12 **return** S

LSH-IS-F (see Algorithm 2) is an evolution of the LSH-IS-S. In this method, one-pass processing is replaced by a more informed selection. The first loop is similar to LSH-IS-S but, instead of directly selecting instances, it first records the bucket to which each instance belongs. When there is only one instance of a class, the instance is rejected, otherwise, if two or more instances of the

[3] The bucket identifier u given to an instance by a family $g \in \mathcal{G}$ can be thought of as the concatenation of all bucket identifiers given by the hash functions in g, since the function families in $\mathcal{G}$ are obtained by using an AND-construction on base functions obtained using Eq. 1. The OR-construction is implemented in the foreach loop at line 3 of Algorithm 1.



(a) Original (b) LSH-IS-S (c) LSH-IS-F

Fig. 4. Example to illustrate the behavior of both algorithms. (a) Initial instances, two buckets are identified by LSH and the line shows the boundary. (b) Instances selected by LSH-IS-S. (c) Instances selected by LSH-IS-F.

same class are present, one of them is randomly chosen. The idea here is to give the algorithm some tolerance of the presence of noise in the input data set.

The execution time of this method is not much larger than in the previous method, since the number of buckets is much lower than the number of instances. Although, the differences in execution time increase with the increase in the number of OR functions.

4.3. Behavior of proposed methods

The aim of this section is to try to shed some light on the behavior of the new algorithms proposed in the paper. As previously stated, LSH-IS-S makes the selection of instances in one pass, selecting one instance of each class in each bucket. On the other hand, LSH-IS-F tries to avoid retaining noisy instances. The more informed selection criterion of LSH-IS-F allows it to remove instances that can be harmful for the classification.

In Fig. 4 we show an example with nine instances: four of one class (crosses) and five of the other (circles). The LSH algorithm is using two buckets, the line represents the boundary between them. LSH-IS-S selects one instance of each class in each bucket (b), while LSH-IS-F does not select the instance of class cross because it is identified as noise.

Fig. 5 illustrates the effect of the algorithms in the XOR data set [42] formed by 400 instances, 200 per class. An outlier was added and highlighted with a gray square. As indicated above, LSH-IS-S retains the instance, while LSH-IS-F removes it.

With the aim of illustrating the behavior of the proposed instance selection methods when the number of hash functions increases, we used the Banana data set. It has two numeric features and two classes, the size of the data set is 5300 instances (see Table 3). Fig. 6(a) shows the original data set. Despite the fact that two clusters can be easily identified, a high overlap exists between the two classes in some regions [21]. Table 2 summarizes the number of instances selected by both algorithms when only one OR function is used and the number of AND functions changes. LSH-IS-F retains less instances, because those instances of one class isolated in one bucket with instances of the other class are identfied as noise and therefore deleted.

Table 2
Number of Banana data set instances that the proposed algorithms retain by the number of AND functions.

Algorithm	Number of AND functions				
	2	4	6	8	10
LSH-IS-S	25	123	249	466	684
LSH-IS-F	24	110	225	423	627

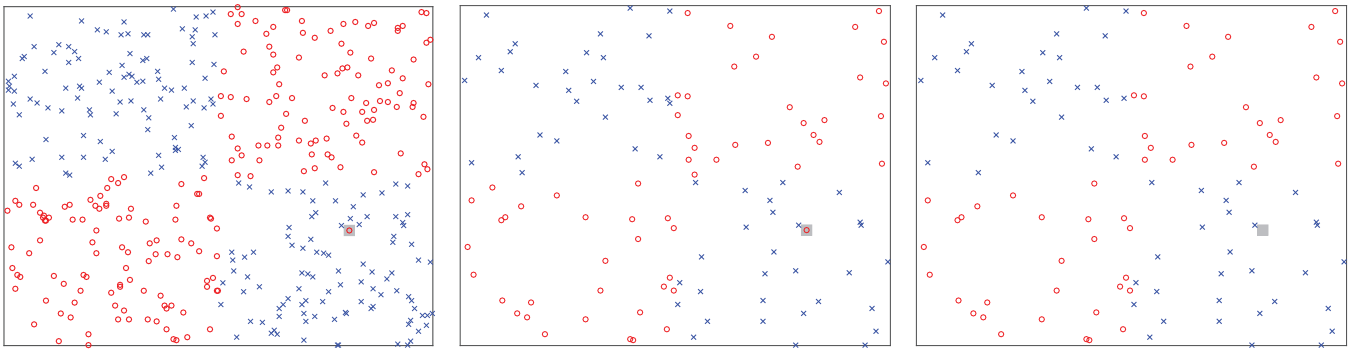
Figs. 6 and 7 show the instances retained by both algorithms, LSH-IS-S and LSH-IS-F respectively, when only one OR function is used and different numbers of AND functions: 2, 4, 6, 8 and 10. The number of retained instances increases with the number of functions. This behavior is quite interesting, because it enables the user to choose whether more or fewer instances are retained, by varying the number of functions that are used.

5. Experimental study

This section presents the experimental study performed to evaluate the new proposed methods. We compared them against seven well-known state-of-the-art instance selection algorithms in a study performed in Weka [62]. The instance selection methods included in the experiments were: CNN, ICF, MSS, DROP3, PSC, HMN-EI, LSBo and the two approaches based on hashing. The parameters selected for the algorithms were those recommended by the authors: the number of nearest neighbors used on ICF and DROP3 were set to $k = 3$, the number of clusters for PSC was set to $6r$ (where r is the number of classes of the data set). Evolutionary algorithms were not included in the experiments, due to their high computational cost.

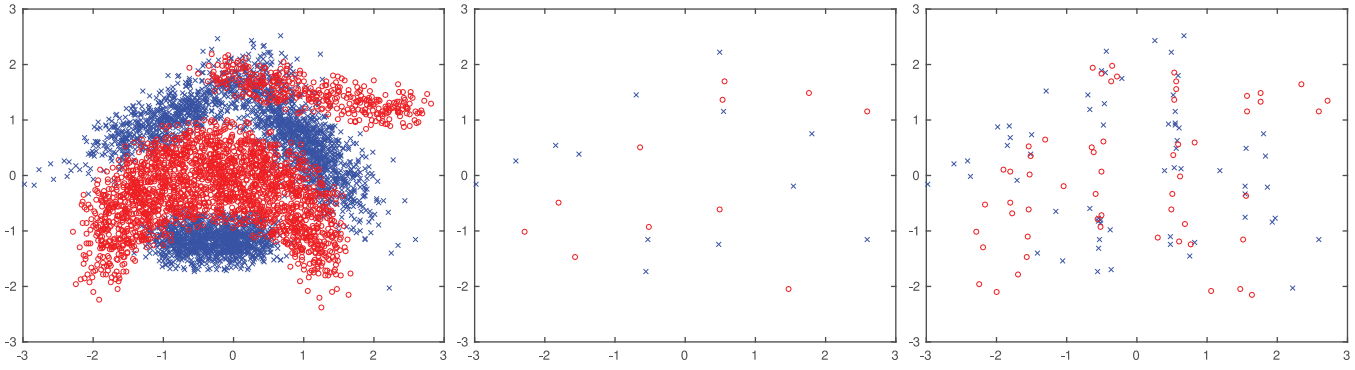
For the experiments, we used 30 data sets from the Keel repository [1] that have at least 1000 instances. Table 3 summarizes the data sets: name, number of features, number of instances and the accuracy given by two classifiers (using ten fold cross-validation): the nearest neighbor classifier with $k = 1$ and the J48, a classifier tree (the Weka implementation of C4.5 [53]). The last five data sets are huge (below dashed line), with more than 299, 000 instances, the traditional instance selection methods are unable to address them. The only transformation carried out was the normalization of all input features, to set their values at between 0 and 1.

We used the nearest neighbor classifier (1NN), as most instance selection methods have been designed for that classifier ($k = 1$ in

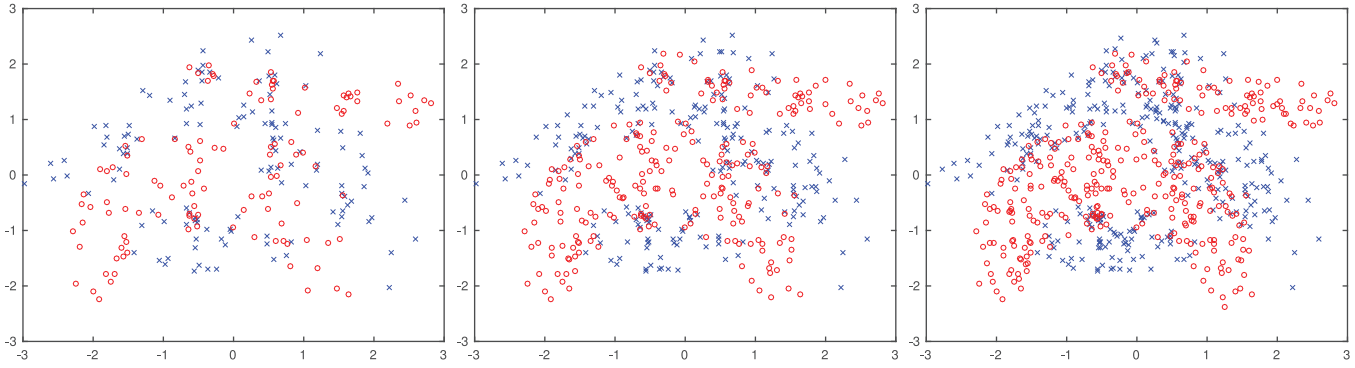


(a) Original (b) LSH-IS-S (c) LSH-IS-F

Fig. 5. (a) Original XOR example, an outlier is highlighted in gray. (b) LSH-IS-S selection maintains the outlier. (c) LSH-IS-F removes the outlier.

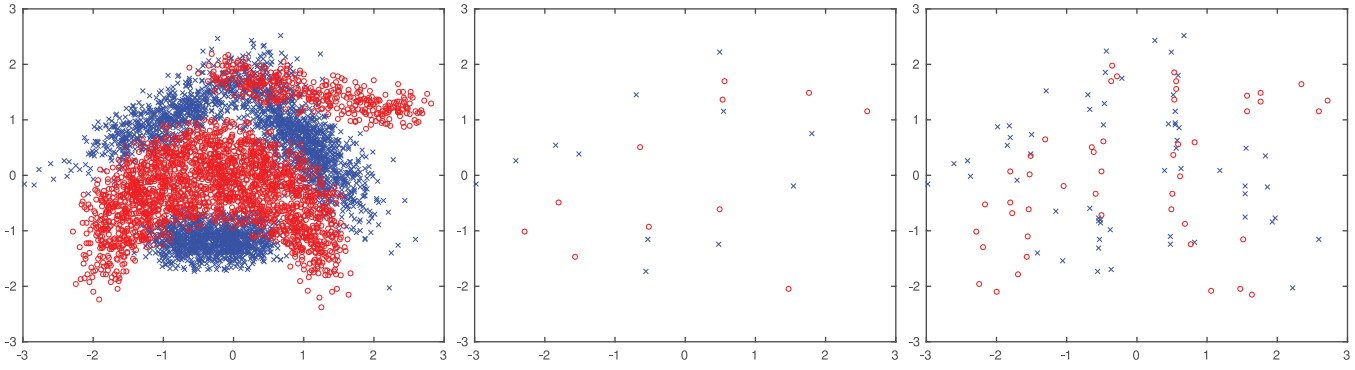


(a) Banana (Orig.) (b) LSH-IS-S: AND=2 (c) LSH-IS-S: AND=4

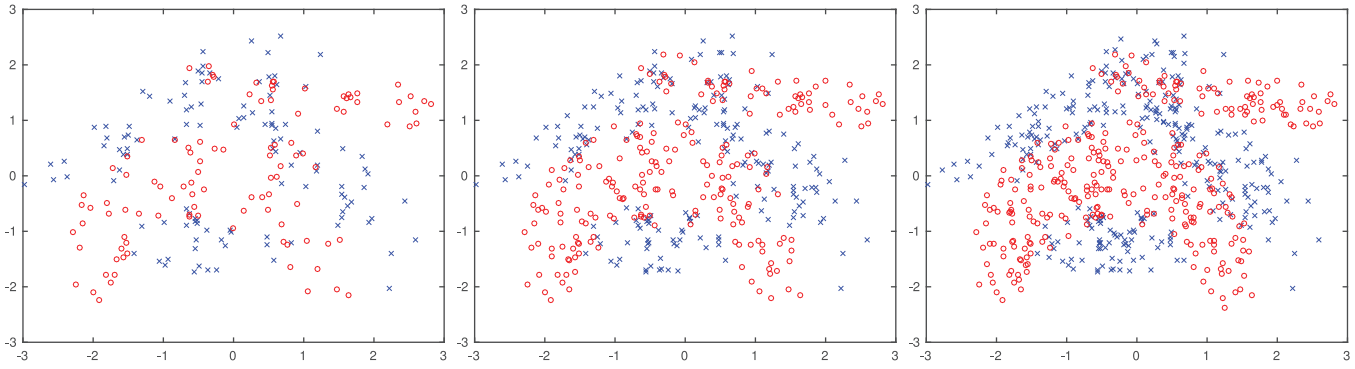


(d) LSH-IS-S: AND=6 (e) LSH-IS-S: AND=8 (f) LSH-IS-S: AND=10

Fig. 6. The number of instances selected by LSH-IS-S as the number of functions increases.



(a) Banana (Orig.) (b) LSH-IS-F: AND=2 (c) LSH-IS-F: AND=4



(d) LSH-IS-F: AND=6 (e) LSH-IS-F: AND=8 (f) LSH-IS-F: AND=10

Fig. 7. The number of instances selected by LSH-IS-F as the number of functions increases.

Table 3
Summary of data sets characteristics: name, number of features, number of instances and accuracy (1NN). Last five data sets, below dashed line, are huge problems.

	Data sets	# attributes		# instances	Accuracy	
		Continuous	Nominal		1NN	J48
1	German	7	13	1000	72.90	71.80
2	Flare	0	11	1066	73.26	73.55
3	Contraceptive	9	0	1473	42.97	53.22
4	Yeast	8	0	1484	52.22	56.74
5	Wine-quality-red	11	0	1599	64.85	62.04
6	Car	0	6	1728	93.52	92.36
7	Titanic	3	0	2201	79.06	79.06
8	Segment	19	0	2310	97.23	96.62
9	Splice	0	60	3190	74.86	94.17
10	Chess	0	35	3196	72.12	81.85
11	Abalone	7	1	4174	19.84	20.72
12	Spam	0	57	4597	91.04	92.97
13	Wine-quality-white	11	0	4898	65.40	58.23
14	Banana	2	0	5300	87.21	89.04
15	Phoneme	5	0	5404	90.19	86.42
16	Page-blocks	10	0	5472	95.91	97.09
17	Texture	40	0	5500	99.04	93.13
18	Optdigits	63	0	5620	98.61	90.69
19	Mushroom	0	22	5644	100.00	100.00
20	Satimage	37	0	6435	90.18	86.28
21	Marketing	13	0	6876	28.74	31.06
22	Thyroid	21	0	7200	92.35	99.71
23	Ring	20	0	7400	75.11	90.95
24	Twonorm	20	0	7400	94.81	85.12
25	Coil 2000	85	0	9822	90.62	93.95
26	Penbased	16	0	10,992	99.39	96.53
27	Nursery	0	8	12,960	98.13	97.13
28	Magic	10	0	19,020	80.95	85.01
29	Letter	16	0	20,000	96.04	87.98
30	KR vs. K	0	6	28,058	73.05	56.58
31	Census	7	30	299,285	92.70	95.42
32	KDDCup99	33	7	494,021	99.95	99.95
33	CovType	54	0	581,012	94.48	94.64
34	KDDCup991M	33	7	1,000,000	99.98	99.98
35	Poker	5	5	1,025,010	50.61	68.25

*k*NN) [37,51]. Moreover, we used the J48 classifier tree, to evaluate the extent to which the instances selected by the algorithms were suitable for training other classifiers.

As shown in Section 3, there are ways of combining the hash functions families that appear more promising than others (see Fig. 3). However, it is unclear which of them will achieve the best results in combination with the proposed algorithms. Therefore, we conducted a study with 60 combinations: AND-constructions, combining between 1 to 10 hash functions, and OR-constructions, combining 1 to 6 functions, obtained by the previous AND-construction, avoiding constructions with too many functions and, consequently, reducing the computational cost.

The subsets selected by the algorithms were used to build a classifier (1NN), the average rank [14] of which was performed over the accuracy of all 60 combinations. Average ranks were calculated as follows: the results of the experiments were sorted, one for the best method, two for the second, and so on. In the case of a tie, values of the ranks were added up and divided into the number of methods that tied. When the ranking of each data set was calculated, the average for each method was computed. Better methods had rankings closer to one. The results of the rankings are shown in Fig. 8. Each cell represents the ranking value for a specific combination of AND-OR-constructions, where the number of functions in the OR-constructions is shown by the *x*-axis and the number of functions in the AND-constructions is shown by the *y*-axis number. The darker the cell the higher its ranking (lower values are better). The best configuration is different for each algorithm:

- LSH-IS-S: the best configuration is one that uses OR-constructions of six functions obtained using an AND-construction on ten functions of the base family (functions obtained using Eq. 1).
- LSH-IS-F: the best results were obtained using OR-constructions of five functions obtained by combining by AND-construction ten functions of the base family.

Fig. 9 shows how the time execution increases, on average, for the proposed algorithms: LSH-IS-S (gray) and LSH-IS-F (black). The higher the number of AND functions, the bigger the gap between LSH-IS-S and LSH-IS-F. This behavior is explained because LSH-IS-F has one loop more than LSH-IS-S (see pseudocodes 1 and 2) that is used to go through all the buckets counting the number of instances of each class. The number of buckets searched increases with the number of hash functions.

Ten fold cross-validation was applied to the instance selection methods under study. The performances were as follows:

- accuracy achieved by 1NN and J48 classifiers trained with the selected subset;
- filtering time by instance selection;
- reduction achieved by instance selection methods (size of the selected subset).

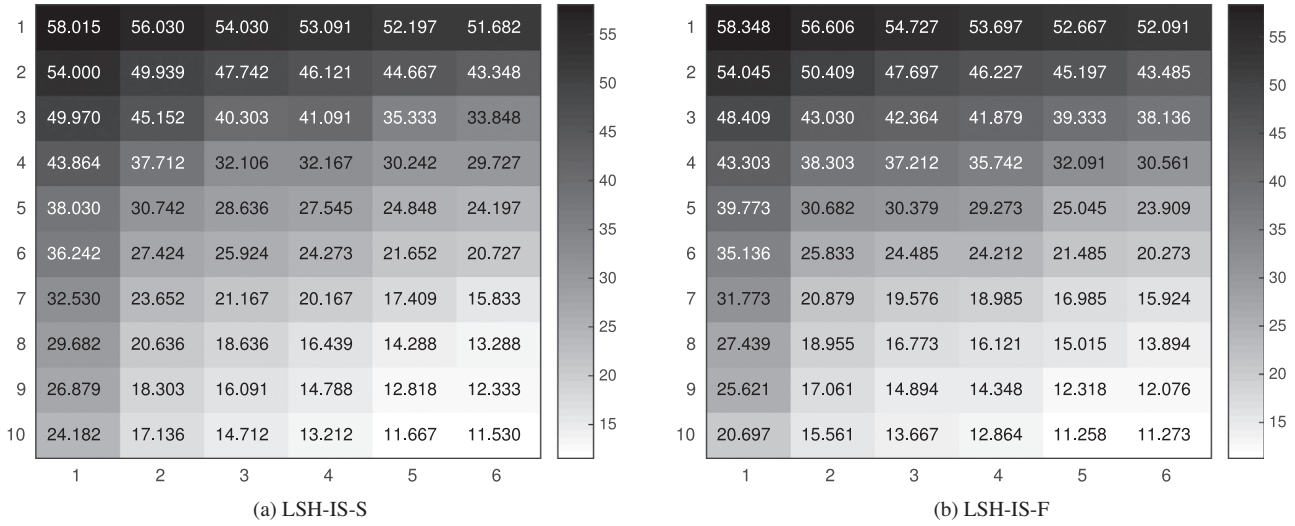


Fig. 8. Average rank over accuracy of the proposed methods for the different configurations of AND-OR constructions. The darker the cell, the higher the ranking (lower is better). Each cell represents an AND-OR-construction where the column is the number of functions in the OR-construction and the row is the number of functions in the AND-construction.

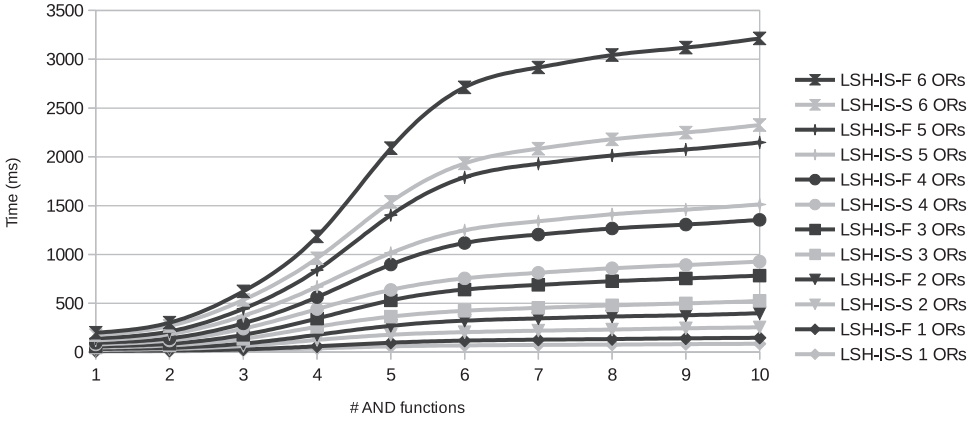


Fig. 9. Average execution time of the proposed algorithms as the number of AND-functions increases. LSH-IS-S is represented in gray and LSH-IS-F in black with different lines and marks for the different numbers of OR-functions.

Table 4
Average ranks and Hochberg procedure over accuracy: 1NN.

Algorithm	Ranking	*p* Hoch.
HMN-EI	2.92	
LSBo	3.85	0.1869
LSH-IS-F	4.45	0.0602
MSS	4.58	0.0553
LSH-IS-S	4.98	0.0139
DROP3	5.03	0.0138
CNN	5.17	0.0088
ICF	5.75	0.0004
PSC	8.27	0.0000

Table 5
Average ranks and Hochberg procedure over accuracy: J48.

Algorithm	Ranking	*p* Hoch.
LSH-IS-F	3.63	
LSH-IS-S	3.88	0.7237
HMN-EI	4.10	0.7237
LSBo	4.57	0.5606
MSS	5.03	0.1909
CNN	5.28	0.0981
ICF	5.67	0.0242
DROP3	5.90	0.0094
PSC	6.93	0.0000

According to the accuracy of 1NN classifier (see Table 4), the best four algorithms were HMN-EI followed by LSBo, LSH-IS-F and MSS. According to the Hochberg procedure [29], differences between them were not significant at a 0.05 significance level. However, differences between LSBo and the other methods were significant. When J48 was used as the classifier (see Table 5), the best six algorithms were LSH-IS-F followed by LSH-IS-S, HMN-EI, LSBo, MSS and CNN; the differences between them were not significant at 0.05. Furthermore, as can be seen, the least accurate model is PSC for both classifiers.

Table 6 shows the average ranks over compression. DROP3 is the best method at a 0.05 significance level. Furthermore, the average reduction rate for each method is also shown. The proposed methods are the most conservative, although, as previously stated, a higher compression could have been achieved using fewer functions in the LSH process.

The third relevant feature of instance selection methods is the time required by the algorithms to calculate the selected subset. Table 7 shows the average rank over execution time of the instance selection algorithms. The three fastest methods were LSH-

Table 6
Average ranks and Hochberg procedure over storage reduction, and average reduction rate.

Algorithm	Ranking	p Hoch.	Reduction rate
DROP3	1.67		0.896
ICF	3.10	0.0427	0.813
LSBo	3.70	0.0081	0.737
PSC	4.70	0.0001	0.762
CNN	5.43	0.0000	0.658
MSS	6.00	0.0000	0.665
HMN-EI	6.10	0.0000	0.577
LSH-IS-F	6.62	0.0000	0.455
LSH-IS-S	7.68	0.0000	0.405

Table 7
Average ranks and Hochberg procedure over filtering time.

Algorithm	Ranking	p Hoch.
LSH-IS-F	1.53	
LSH-IS-S	1.57	0.9624
PSC	3.00	0.0761
MSS	4.20	0.0005
ICF	5.33	0.0000
LSBo	6.73	0.0000
HMN-EI	6.70	0.0000
DROP3	7.67	0.0000
CNN	8.27	0.0000

IS-F, LSH-IS-S and PSC, between them differences were not significant at a 0.05 significance level. The differences were significant from MSS and the following methods; the slowest was CNN. It is worth noting that PSC achieved, according to the significance tests, a really competitive execution time. Nevertheless the shortcoming of PSC was its poor accuracy, as it obtained the worst results of all of the methods under analysis, as noted in Tables 4 and 5.

It might be surprising that LSH-IS-F was faster than LSH-IS-S, because this contradicts Fig. 9. The reason for this was the number of functions used by the algorithms; as commented on at the beginning of this section, LSH-IS-S was launched using six OR functions, while LSH-IS-F was launched with only five.

Fig. 10 shows the filtering time as a function of the number of instances. The results obtained with the data sets of the experiments were used to draw these figures. Since there were no results available for all possible values of numbers of instances, the available results were used to draw Bezier lines and to show the general trend of the algorithms. Although other methods (CNN, HMN-EI, LSBo, DROP3, MSS and ICF) have an execution time that increases swiftly, our algorithms based on hashing are at the bottom of the figures together with PSC. However, in the logarithmic scale, the growth of PSC is visibly greater.

As a summary of the experimentation with medium size data sets, we can highlight that the proposed methods achieved competitive results in terms of accuracy. Considering the reduction rate, DROP3 achieved the maximum compression, while our methods were the worst in terms of compression. Finally, considering execution time, the methods presented in the paper were able to compute the selected subset much faster than the other algorithms in the state of the art. Exceptionally, PSC worked surprisingly fast, although slower than the speed of our proposals, and with the shortcoming of the poor accuracy when its selected subsets were used for training the classifiers.

5.1. Huge problems

Due to the fact that instance selection methods are not able to face huge problems, the experimental study performed over Census, CovType, KDDCup99, KDDCup991M[4] and Poker (see Table 3), the algorithms proposed were tested against the Democratic Instance Selection (DIS) [22]. Although PSC showed a competitive results in terms of execution time, it was not included in the study of huge problems, because of its poor accuracy.

Table 8
Average ranks over accuracy for huge problems.

Algorithm	Ranking
LSH-IS-F	2.0
DIS.RNN	2.2
LSH-IS-S	2.6
DIS.DROP3	3.6
DIS.ICF	4.6

Table 9
Average ranks over storage reduction for huge problems.

Algorithm	Ranking
DIS.RNN	1.8
DIS.DROP3	2.4
LSH-IS-F	3.2
DIS.ICF	3.4
LSH-IS-S	4.2

As in the previous experiments, ten fold cross-validation was performed on LSH-IS-S and LSH-IS-F in Weka. Testing error, using 1-NN classifier, and storage reduction were reported and compared against results published in [22]. Execution times were not compared because different implementations and machines would not have allowed a fair comparison.

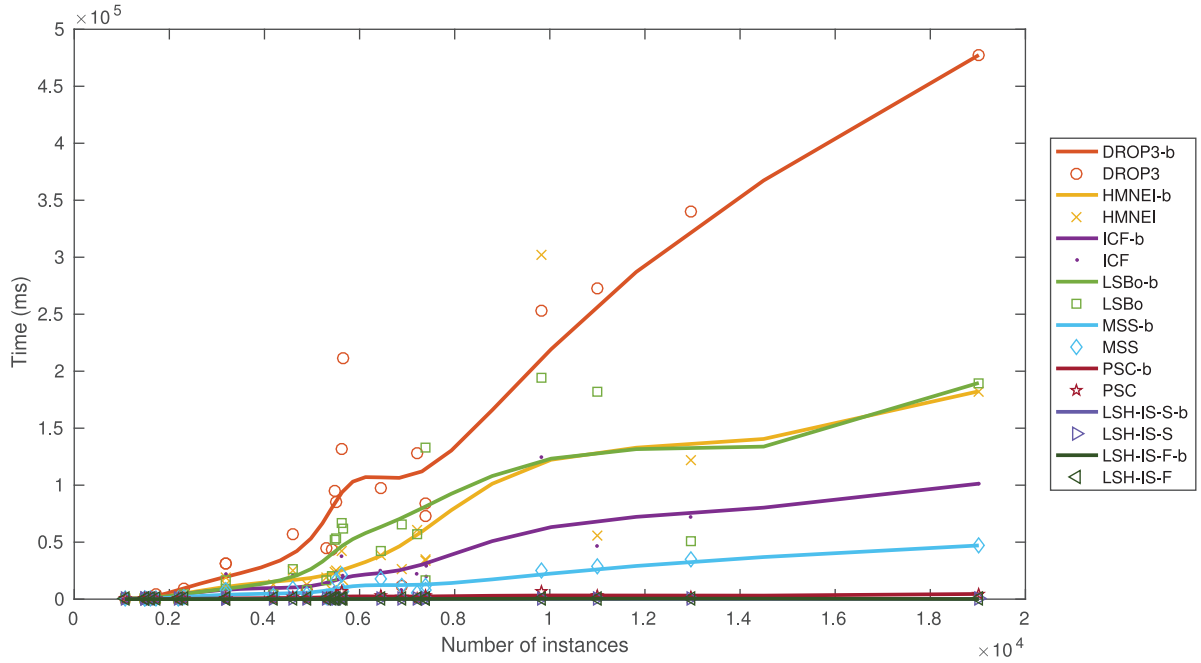
The main conclusion of the experiments was that our methods can face huge problems. Results of average ranks over the accuracy are shown in Table 8. The accuracy of the methods under study is similar to DIS, though the most accurate method is LSH-IS-F. As in the medium size experiment, LSH-IS-F improved the accuracy with regard to LSH-IS-S. On the other hand, the Table 9 shows the average ranks over storage reduction. In terms of compression, the best method was DIS.RNN, as proved in medium size data sets, while the methods based on hashing were too conservative. However, the number of instances that they retain can be adjusted by the number of functions used. The success of the proposed methods is even more remarkable when compared against scalable approaches. The simple idea of using LSH overcomes the democratization methods and opens the way to their use in huge data sets and big data.

6. Conclusions

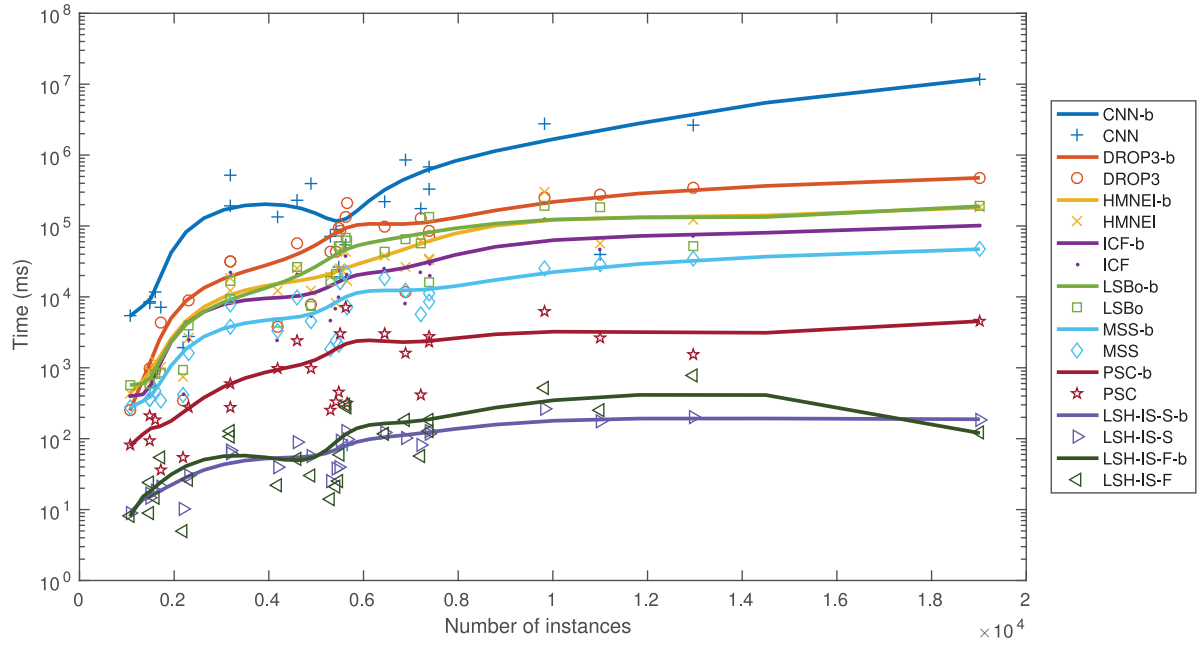
The paper has introduced a novel approach to the use of families of locality sensitive functions (LSH) for instance selection. Using this approach, two new algorithms of linear complexity have been designed. In one approach, the data are processed in one pass, which allows the algorithm to make the selection without requiring that the whole data set to fit in memory. The other approach needs two passes: one processes each instance of the data set, and the second processes the buckets of the families of hash functions. Their speed and low memory consumption mean that they are suitable for big data processing.

The experiments have shown that the strength of our methods is the speed, which is achieved through a small decrease in accuracy and, more remarkably, the reduction rate. Although the

[4] We divided the original KDDCup 1999 data set into two sets with different number of instances: KDDCup99 has 10% of the original instances and KDDCup991M has a million.



(a) Linear scale



(b) Logarithmic scale

Fig. 10. Computational cost of tested methods, in linear (a) and logarithmic scale (b) on the *y* axis. In (a) the CNN was not plotted because its growth was so high that it was not possible to appreciate the differences between the other methods. The dots are the results on the available data sets, lines (denoted by "–b" in the legend) are the Bezier lines built with these dots to show the general trend of the algorithms.

best methods according to accuracy differ depending on the classifier that is used, the proposed methods offer a competitive performance. Moreover, the reduction rate can be adjusted by increasing or by decreasing the number of hash functions that are used.

Furthermore, the proposed methods were evaluated on huge problems and compared against Democratic Instance Selection, a linear complexity method. Experimental results on accuracy showed how our methods outperformed DIS, even though our methods were conceptually much simpler.

7. Future work

In their current version, the way the algorithms make the instance selection is very simple and quite "naive". The selected subset could be improved using additional information about the instances assigned to each bucket, and not just the count of instances of each class. A future research line could be to store additional information of the instances assigned to each bucket: for example, simple statistics such as the incremental average of instances

in the bucket, or the percentage of instances of each class in the bucket. This information might mean that the instance selection process would be better informed, without excessively penalizing run-time. Although prototype generation has not been analyzed in the paper, the generation of a new instance, or group of them, for each bucket is one of the future lines of research. This idea can be developed using LSH-IS-F, seeking each of the buckets to build or to create a new set of instances, by selecting the medoids or centroids of the instances in the buckets.

According to [63], one of the most challenging problems in data mining research is mining data streams in extremely large databases. Accurate and fast processes able to work on stream are required, without any assumption that information can be stored in large databases and repeatedly accessed. One of the problems that arises in those environments is called *concept drift*, which appears when changes in the context take place. In the management of concept drift, three basic approaches can be distinguished: ensemble learning, instance weighting and instance selection [59]. A comparison of the proposed method in a streaming benchmark would be made to test whether LSH-IS-S can beat the state-of-the-art algorithms that are able to deal in streaming data [18].

Many more research approaches can be considered, but the principal one for us is to adapt the new methods to a big data environment. We are working on the adaptation of this idea to a MapReduce framework, which offers a robust environment to face up to the processing of huge data sets over clusters of machines [58].

Acknowledgements

Supported by the Research Projects TIN 2011-24046 and TIN 2015-67534-P from the Spanish Ministry of Economy and Competitiveness.

References

[1] J. Alcalá-Fdez, L. Sánchez, S. Garcia, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J. Fernández, F. Herrera, Keel: a software tool to assess evolutionary algorithms for data mining problems, Soft Comput. 13 (3) (2009) 307–318, doi:10.1007/s00500-008-0323-y.

[2] Álvar Arnaiz-González, J.F. Díez-Pastor, J.J. Rodríguez, C.I. García-Osorio, Instance selection for regression by discretization, Expert Syst. Appl. 54 (2016) 340–350, doi:10.1016/j.eswa.2015.12.046.

[3] A. Andoni, P. Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, Commun. ACM 51 (1) (2008) 117–122, doi:10.1145/1327452.1327494.

[4] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (6) (1998) 891–923, doi:10.1145/293347.293348.

[5] D. Asimov, The grand tour: A tool for viewing multidimensional data, SIAM J. Sci. Stat. Comput. 6 (1) (1985) 128–143, doi:10.1137/0906011.

[6] R. Barandela, F.J. Ferri, J.S. Sánchez, Decision boundary preserving prototype selection for nearest neighbor classification, IJPRAI 19 (6) (2005) 787–806.

[7] H. Brighton, C. Mellish, On the consistency of information filters for lazy learning algorithms, in: J. Żytkow, J. Rauch (Eds.), Principles of Data Mining and Knowledge Discovery, Lecture Notes in Computer Science, volume 1704, Springer Berlin Heidelberg, 1999, pp. 283–288, doi:10.1007/978-3-540-48247-5_31.

[8] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Min. Know. Discov. 6 (2) (2002) 153–172, doi:10.1023/A:1014043630878.

[9] J. Calvo-Zaragoza, J.J. Valero-Mas, J.R. Rico-Juan, Improving kNN multi-label classification in prototype selection scenarios using class proposals, Pattern Recognit. 48 (5) (2015) 1608–1622, doi:10.1016/j.patcog.2014.11.015.

[10] J.R. Cano, F. Herrera, M. Lozano, Stratification for scaling up evolutionary prototype selection, Pattern Recognit. Lett. 26 (7) (2005) 953–963, doi:10.1016/j.patrec.2004.09.043.

[11] T. Cover, P. Hart, Nearest neighbor pattern classification, Inf. Theory IEEE Trans. 13 (1) (1967) 21–27, doi:10.1109/TIT.1967.1053964.

[12] M. Datar, N. Immorlica, P. Indyk, V.S. Mirrokni, Locality-sensitive hashing scheme based on *p*-stable distributions, in: Proceedings of the Twentieth Annual Symposium on Computational Geometry, in: SCG '04, ACM, New York, NY, USA, 2004, pp. 253–262, doi:10.1145/997817.997857.

[13] A. de Haro-García, N. García-Pedrajas, A divide-and-conquer recursive approach for scaling up instance selection algorithms, Data Min. Knowl. Discov. 18 (3) (2009) 392–418, doi:10.1007/s10618-008-0121-2.

[14] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.

[15] J. Derrac, S. García, F. Herrera, Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability, Memetic Comput. 2 (3) (2010) 183–199, doi:10.1007/s12293-010-0048-1.

[16] J. Derrac, S. García, F. Herrera, A survey on evolutionary instance selection and generation, Int. J. Appl. Metaheuristic Comput. 1 (1) (2010) 60–92, doi:10.4018/jamc.2010102604.

[17] J. Derrac, N. Verbiest, S. García, C. Cornelis, F. Herrera, On the use of evolutionary feature selection for improving fuzzy rough set based prototype selection, Soft Comput. 17 (2) (2013) 223–238, doi:10.1007/s00500-012-0888-3.

[18] G. Ditzler, M. Roveri, C. Alippi, R. Polikar, Learning in nonstationary environments: a survey, Comput. Intell. Mag., IEEE 10 (4) (2015) 12–25.

[19] F. Dornaika, I.K. Aldine, Decremental sparse modeling representative selection for prototype selection, Pattern Recognit. 48 (11) (2015) 3714–3727, doi:10.1016/j.patcog.2015.05.018.

[20] R.O. Duda, P.E. Hart, D.G. Stork, Pattern classification, John Wiley & Sons, New York City, United States, 2012.

[21] S. Garcia, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: Taxonomy and empirical study, Pattern Anal. Mach. Intell. IEEE Trans. 34 (3) (2012) 417–435, doi:10.1109/TPAMI.2011.142.

[22] C. García-Osorio, A. de Haro-García, N. García-Pedrajas, Democratic instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts, Artif. Intell. 174 (5-6) (2010) 410–441, doi:10.1016/j.artint.2010.01.001.

[23] N. García-Pedrajas, A. de Haro-García, Boosting instance selection algorithms, Knowl. Based Syst. 67 (2014) 342–360, doi:10.1016/j.knosys.2014.04.021.

[24] N. García-Pedrajas, J.A. Romero del Castillo, D. Ortiz-Boyer, A cooperative co-evolutionary algorithm for instance selection for instance-based learning, Mach. Learn. 78 (3) (2010) 381–420, doi:10.1007/s10994-009-5161-3.

[25] G. Gates, The reduced nearest neighbor rule (corresp.), Inf. Theor. IEEE Trans. 18 (3) (1972) 431–433, doi:10.1109/TIT.1972.1054809.

[26] A. Guillen, L. Herrera, G. Rubio, H. Pomares, A. Lendasse, I. Rojas, New method for instance or prototype selection using mutual information in time series prediction, Neurocomputing 73 (10-12) (2010) 2030–2038, doi:10.1016/j.neucom.2009.11.031. Subspace Learning / Selected papers from the European Symposium on Time Series Prediction.

[27] P. Hart, The condensed nearest neighbor rule (corresp.), Inf. Theor. IEEE Trans. 14 (3) (1968) 515–516.

[28] P. Hernandez-Leal, J. Carrasco-Ochoa, J. Martínez-Trinidad, J. Olvera-López, Instance rank based on borders for instance selection, Pattern Recognit. 46 (1) (2013) 365–375, doi:10.1016/j.patcog.2012.07.007.

[29] Y. Hochberg, A sharper bonferroni procedure for multiple tests of significance, Biometrika 75 (4) (1988) 800–802, doi:10.1093/biomet/75.4.800.

[30] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, in: STOC '98, ACM, New York, NY, USA, 1998, pp. 604–613, doi:10.1145/276698.276876.

[31] N. Jankowski, M. Grochowski, Comparison of instances seletion algorithms I. algorithms survey, in: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L. Zadeh (Eds.), Artificial Intelligence and Soft Computing - ICAISC 2004, Lecture Notes in Computer Science, volume 3070, Springer Berlin Heidelberg, 2004, pp. 598–603, doi:10.1007/978-3-540-24844-6_90.

[32] S.-W. Kim, J.B. Oommen, A brief taxonomy and ranking of creative prototype reduction schemes, Pattern Anal. Appl. 6 (3) (2003) 232–244, doi:10.1007/s10044-003-0191-0.

[33] M. Kordos, S. Białka, M. Blachnik, Instance selection in logical rule extraction for regression problems, in: L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L.A. Zadeh, J.M. Zurada (Eds.), Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science, volume 7895, Springer Berlin Heidelberg, 2013, pp. 167–175, doi:10.1007/978-3-642-38610-7_16.

[34] E. Kushilevitz, R. Ostrovsky, Y. Rabani, Efficient search for approximate nearest neighbor in high dimensional spaces, in: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, in: STOC '98, ACM, New York, NY, USA, 1998, pp. 614–623, doi:10.1145/276698.276877.

[35] J. Leskovec, A. Rajaraman, J.D. Ullman, Mining of Massive Datasets, third, Cambridge University Press, Cambridge, United Kingdom, 2014.

[36] E. Leyva, A. González, R. Pérez, Knowledge-based instance selection: A compromise between efficiency and versatility, Knowl. Based Syst. 47 (2013) 65–76, doi:10.1016/j.knosys.2013.04.005.

[37] E. Leyva, A. González, R. Pérez, Three new instance selection methods based on local sets: a comparative study with several approaches from a bi-objective perspective, Pattern Recognit. 48 (4) (2015) 1523–1537, doi:10.1016/j.patcog.2014.10.001.

[38] J. Li, Y. Wang, A new fast reduction technique based on binary nearest neighbor tree, Neurocomputing 149, Part C (2015) 1647–1657, doi:10.1016/j.neucom.2014.08.028.

[39] T. Liu, A.W. Moore, K. Yang, A.G. Gray, An investigation of practical approximate nearest neighbor algorithms, in: Advances in NeuralInformation Processing Systems, MIT Press, 2004, pp. 825–832.

[40] A. Lumini, L. Nanni, A clustering method for automatic biometric template selection, Pattern Recognit. 39 (3) (2006) 495–497, doi:10.1016/j.patcog.2005.11.004.

[41] E. Marchiori, Hit miss networks with applications to instance selection, J. Mach. Learn. Res. 9 (2008) 997–1017.
[42] E. Marchiori, Class conditional nearest neighbor for large margin instance selection, IEEE Trans. Pattern Anal. Mach. Intell. 32 (2) (2010) 364–370, doi:10.1109/TPAMI.2009.164.
[43] E. Merelli, M. Pettini, M. Rasetti, Topology driven modeling: the is metaphor, Natural Comput. 14 (3) (2014) 421–430, doi:10.1007/s11047-014-9436-7.
[44] C. Moretti, K. Steinhaeuser, D. Thain, N.V. Chawla, Scaling up classifiers to cloud computers, in: Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on, 2008, pp. 472–481, doi:10.1109/ICDM.2008.99.
[45] L. Nanni, A. Lumini, Prototype reduction techniques: A comparison among different approaches, Expert Syst. Appl. 38 (9) (2011) 11820–11828, doi:10.1016/j.eswa.2011.03.070.
[46] J. Olvera-López, J. Carrasco-Ochoa, J. Martínez-Trinidad, A new fast prototype selection method based on clustering, Pattern Anal. Appl. 13 (2) (2009) 131–141, doi:10.1007/s10044-008-0142-x.
[47] J. Olvera-López, J. Martínez-Trinidad, J. Carrasco-Ochoa, J. Kittler, Prototype selection based on sequential search, Intell. Data Anal. 13 (4) (2009) 599–631.
[48] S. Ougiaroglou, G. Evangelidis, D.A. Dervos, FHC: an adaptive fast hybrid method for *k*-NN classification, Logic J. IGPL (2015).
[49] D. Peralta, S. del Río, S. Ramírez-Gallego, I. Triguero, J.M. Benitez, F. Herrera, Evolutionary feature selection for big data classification: a mapreduce approach, Math. Probl. Eng. 501 (2015) 246139.
[50] J. Pérez-Benítez, J. Pérez-Benítez, J. Espina-Hernández, Novel data condensing method using a prototype's front propagation algorithm, Eng. Appl. Artif. Intell. 39 (2015) 181–197.
[51] J. Pérez-Rodríguez, A.G. Arroyo-Peña, N. García-Pedrajas, Simultaneous instance and feature selection and weighting using evolutionary computation: Proposal and study, Appl. Soft Comput. 37 (2015) 416–443, doi:10.1016/j.asoc.2015.07.046.
[52] E. Pękalska, R.P. Duin, P. Paclík, Prototype selection for dissimilarity-based classifiers, Pattern Recognit. 39 (2) (2006) 189–208, doi:10.1016/j.patcog.2005.06.012. Part Special Issue: Complexity ReductionPart Special Issue: Complexity Reduction.
[53] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
[54] G. Ritter, H. Woodruff, S. Lowry, T. Isenhour, An algorithm for a selective nearest neighbor decision rule, IEEE Trans. Inf. Theor. 21 (6) (1975) 665–669.
[55] M.B. Stojanović, M.M. Božić, M.M. Stanković, Z.P. Stajić, A methodology for training set instance selection using mutual information in time series prediction, Neurocomputing 141 (2014) 236–245, doi:10.1016/j.neucom.2014.03.006.
[56] H. Tran-The, V.N. Van, M.H. Anh, Fast approximate near neighbor algorithm by clustering in high dimensions, in: Knowledge and Systems Engineering (KSE), 2015 Seventh International Conference on, 2015, pp. 274–279, doi:10.1109/KSE.2015.72.
[57] I. Triguero, J. Derrac, S. Garcia, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification, Syst. Man Cybern. Part C 42 (1) (2012) 86–100, doi:10.1109/TSMCC.2010.2103939.
[58] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, MRPR: A mapreduce solution for prototype reduction in big data classification, Neurocomputing 150, Part A (2015) 331–345, doi:10.1016/j.neucom.2014.04.078.
[59] A. Tsymbal, The problem of concept drift: definitions and related work, Comput. Sci. Department, Trinity College Dublin 106 (2004) 2.
[60] D.R. Wilson, T.R. Martinez, Instance pruning techniques, in: Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97), Morgan Kaufmann, 1997, pp. 404–411.
[61] D.R. Wilson, T.R. Martinez, Reduction techniques for instance-based learning algorithms, Mach. Learn. 38 (3) (2000) 257–286, doi:10.1023/A:1007626913721.
[62] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, third, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.
[63] Q. Yang, X. Wu, 10 challenging problems in data mining research, Int. J. Inf. Technol. Decis. Mak. 05 (04) (2006) 597–604, doi:10.1142/S0219622006002258.